Journal of Engineering Science and Technology
Vol. 9, No. 4 (2014) 432 - 442

DEVELOPMENT OF A NEW ALGORITHM FOR KEY AND S-BOX GENERATION IN BLOWFISH ALGORITHM

TAYSEER S. ATIA

College of Software Engineering and Networking, the Iraqia University, Baghdad, Iraq
E-mail: tayseer_salman@yahoo.com

Abstract

Blowfish algorithm is a block cipher algorithm, its strong, simple algorithm used to encrypt data in block of size 64-bit. Key and S-box generation process in this algorithm require time and memory space the reasons that make this algorithm not convenient to be used in smart card or application requires changing secret key frequently. In this paper a new key and S-box generation process was developed based on Self Synchronization Stream Cipher (SSS) algorithm where the key generation process for this algorithm was modified to be used with the blowfish algorithm. Test result shows that the generation process requires relatively slow time and reasonably low memory requirement and this enhance the algorithm and gave it the possibility for different usage.

Keywords: SSS, Ciphertext, RB, LFSR.

1. Introduction

Due to the incremental growth of the internet and communication networks the security and privacy of users are threatened and become exposed. The block cipher algorithms are asymmetric key cryptography which is developed for providing encrypts data with reasonable amount of time and space.

Blowfish is a block cipher that encrypts data in 8-byte blocks. The algorithm consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts a variable-length key of at most 64 bytes (512 bits) into several subkey arrays totalling 4168 bytes [1]. Data encryption occurs via 16 rounds. Each round consists of a key-dependent permutation and a key- and data-dependent substitution. The fundamental operations were chosen with speed in mind. XOR, ADD, and MOV from a cache are efficient on both Intel and Motorola architectures [1, 2].

Nomenclatures

K_i	Key
P	P-array
S	s-box
T	Temporary result
V_t	key stream
X	Input byte

Greek Symbols

σ_t	State word

Abbreviations

ISRC	Information security research centre
LFSR	Linear feedback shift register
NLF	Nonlinear function
SRB	Saturated resilient boolean
SSS	Self synchronisation stream cipher

There is a trade-off between the memory requirements and the algorithm speed. Faster operation in encryption needs precomputation of the subkeys which need a large memory system [2]. Approximately 4 kB is required which made the algorithm inapplicable in smart card application. Computing the subkeys using the algorithm every time results in slower operation which made the algorithm inefficient to use in application requires changing secret key frequently.

SSS is a self-organization stream cipher developed from SOBER [3, 4] which proposed by Rose in1998. This cipher is designed to generate secret key up to 128 bits. It also offers message encryption, message integrity or both. SSS can be used in smart cards or large computers [5].

2. Description of Used Material

This section includes the complete description of blowfish algorithm and the SSSS stream cipher.

2.1. Blowfish description

Blowfish use two primitive operation, addition of two words performed modulo 2^{32}, bitwise exclusive-OR. The plain text is divided into two 32-bit halves LE_0 and RE_0 and after 16 round the LE_{17} and RE_{17} is combined to form 64 ciphertext. Decryption is exactly the same as encryption, except that P1, P2,… P18 are used in the reverse order [4]. Figure 1(a) shows the encryption pseudo code and Fig. 1(b) shows the decryption pseudo code [5].

2.1.1. Function *F*

This function divides input into four eight-bit quarters: *a*, *b*, *c*, and *d*. Then,

$F[a, b, c, d] = ((S1, a + S2, b \bmod 232)\ XOR\ S3,c) + S4,d \bmod 232.$

The function that combines the four S-box outputs is as fast as possible. Figure 2 shows this function [5, 6].

For i= 1 to 16

$RE_i = LE_i$ XOR P_i
$LE_i = F\ [RE_{i-1}]$ XOR RE_{i-1}
$LE_{17} = RE_{16}$ XOR P_{18}
$RE_{17} = LE_{16}$ XOR P_{17}

(a) Blowfish Encryption Pseudo Code.

For i= 1 to 16

$RD_i = LD_{i-1}$ XOR P_{19-i}
$LD_i = F\ [RD_{i-1}]$ XOR RD_{i-1}
$LD_{17} = RD_{16}$ XOR P_1
$RD_{17} = LD_{16}$ XOR P_2

(b) Blowfish Decryption Pseudo Code.

Fig. 1. Blowfish Encryption and Decryption.

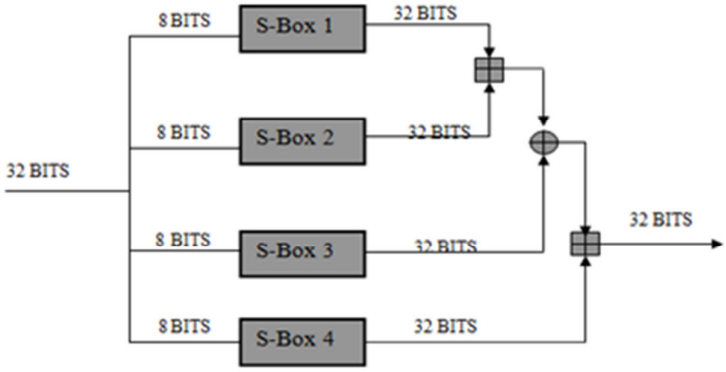

Fig. 2. Function *F* of the Blowfish Algorithm.

2.1.2. Subkeys generation

Blowfish uses a large number of subkeys. These keys must be precomputed before any data encryption or decryption. The P-array consists of 18 32-bit subkeys: P1, P2… P18. There are also four 32-bit S-boxes with 256 entries each [5]:

S1, 0, S1, 1… S1, 255;
S2, 0, S2, 1… S2, 255;
S3, 0, S3, 1… S3, 255;
S4, 0, S4, 1… S4, 255;

The S-box and P-array generation process works as follow [5, 7]:

1. Initialize first the P-array and then the four S-boxes, in order, with a fixed string. This string consists of the hexadecimal digits of π:-

   P1 = 0x243f6a88,
   P2 = 0x85a308d3,
   P3 = 0x13198a2e,
   P4 = 0x03707344, etc.

2. XOR P1 with the first 32 bits of the key, XOR P2 with the second 32-bits of the key, and so on for all bits of the key (possibly up to P16). Repeatedly cycle through the key bits until the entire P-array has been XORed with key bits.
3. Encrypt the all-zero string with the Blowfish algorithm, using the subkeys.
4. Replace P1 and P2 with the output of step (3).
5. Encrypt the output of step (3) using the Blowfish algorithm with the modified subkeys.
6. Replace P3 and P4 with the output of step (5).
7. Continue the process, replacing all entries of the P-array, and then all four S-boxes in order, with the output of the continuously changing Blowfish algorithm.

In total, 521 iterations are required to generate all required subkeys.

2.2. SSS description

Key stream generation based on simple XOR operation and modulo 2^{16}. The generator is composed of a simple shift register and non-liner filter. At time t, the generator outputs a vector of words (each is of 16-bit block) called state $\sigma_t = (r_t[0], \ldots, r_t[16])$. The next state σ_{t+1} can be generated each time using the following process [4]:

1. $r_t[0]$ is abandoned
2. $r_{t+1}[\mathrm{i}] = r_t[i+1]$, for $i = 0..15$
3. $r_{t+1}[16] = c_t$
4. $r_{t+1}[14] = r_t + 1[14] + f(c_t >>> 8)$
5. $r_{t+1}[12] = f(r_t + 1[12])$
6. $r_{t+1}[1] = r_t + 1[1] >>> 8$

After generate the successive states, they forwarded to non-linear filter to produce 16-bit key stream words denoted v_t and it's obtained as [4]:

$v_t = \mathrm{NLF}(\sigma_t) = f(f(r[0] + r[16]) + r[1] + r[6] + r[13]) >>> 8)$ XOR $r[0]$.

The function f in fact is an 8×16 bit S-box defined as $f(a) = \mathrm{SBox}[a_H]$ XOR a [4]. The S-box is a combination of skipjack S-box [6] and Qbox S-box [8]. Each S-box is represented as a table of 256 entries in hex. The following pseudocode, Fig. 3, demonstrates how the S-box is generated [4].

```
WORD Sbox (UCHAR *key, int keylen, WORD w)
{Register int i; WORD t; UCHAR b;
t = 0;
b = HIGHBYTE (w);
For (i = 0; i < keylen; ++i) {
b = ftable [b ^ key[i]]; "ftable is the skipjack s-box"
t ^= ROTL(Qbox[b], i); }
Return ((b << (WORDBITS-8)) | (t & LOWMASK)) ^ (w & LOWMASK) ;}
```

Fig. 3. S-box Pseudo Code.

3. The Proposed Method

This process consists of the following subprocess: 1. Initialization, 2. P-array generation, 3. S-box generation. P-array and S-box generation process are an extended to the SSS key stream generation process with some important modification to make it compatible with the requirement of blowfish key generation and S-box generation process.

• **Initialization:** This process initializes the following data structures:

1. Keymatrix: is a two dimension array of size 4×4 each of 16-bit, this matrix represents the key of length 128-bit used to build key-dependent S-box in SSS.
2. Lr: is a one-dimension array consists of 18 elements each of 32-bits; it represents the state register in SSS and used to generate P-array.
3. Temp: used to generate P-array and its designed to carry values necessary to feed the Lr register.

All these data structures are initialized using secret nonce (a number used only once) and a resilient Boolean function of (6, 2, 3, 24) reads as 2-resilent function, 6-variables, nonlinearity 24 and algebraic degree 3. Any function can be used with different parameters but for reasonable time and security this function is selected.

$f = x_2 \; x_4 \; x_5 \; XOR \; x_1 x_4 \; XOR \; x_1 x_5 \; XOR \; x_2 x_5 \; XOR \; x_3 x_5 \; XOR \; x_1 \; XOR \; x_2 \; XOR \; x_3 \; XOR \; x_6$

Figure 4 shows the LFSR structure used in this function. The length of the LFSRs used are (7, 9, 11, 13, 21, 23) with linear feedback function between cell 0 and cell *n*-1 from each register.

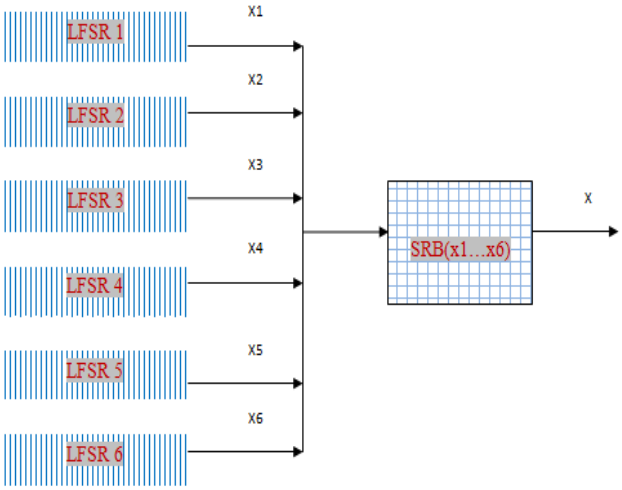

Fig. 4. Structure of Binary Sequence for SRB of Six Variables.

• **P-array generation**:

In blowfish, P-array is an array of size 18 elements each with 32 bit. This array will be generated using ***r*** register in SSS with a small modification to the size of

the register and its pseudo code. Figure 5 shows the diagram of the overall generation process.

The following pseudo code in vb.net describes this process

```
 For i=0 to 17
  Vt=f(f(lr(0)+lr(17))+ lr(1)+lr(7)+lr(14))>>8
P(i)= vt xor temp(i)
  For j=0 to 16
      Lr(j)=lr(j+1)
Next
lr(17) = p(i)
        lr(15) = lr(15) + f(p(i) >> 8)
        lr(13) = f(lr(13))
        lr(1) = lr(1) >> 8
      Next
```

F is defined as:
F(a)= sbox(keymatrix, j, a) xor a, J value must be 2-bit length to generate values in range (0,1,2,3) only.

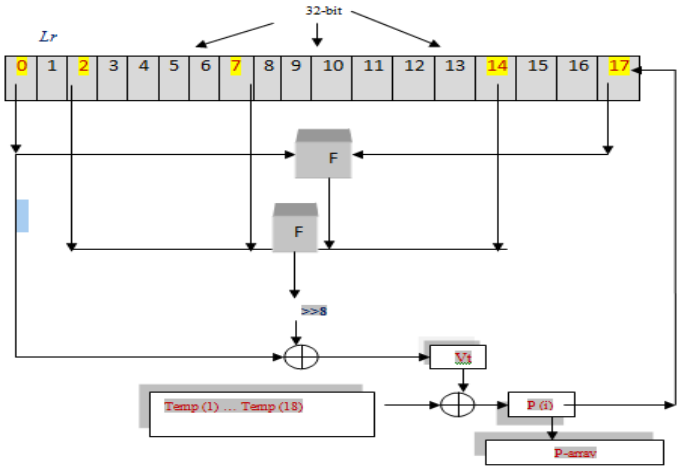

Fig. 5. Overall Generation Process.

- **S-box generation:**

This process uses skipjack table and Qbox table found in SSS and the same generation process with some changes to the value of the key used in SSS. This value is treated as a two dimension array of size 4×4 each of 16-bit and this change is necessary to generate four s-boxes in blowfish rather than one S-box in SSS also the output from this process is 32-bit value instead of 16 values. The following pseudo code in vb.net displays single S-box generation:

```
Function sbox (keymatrix(4,4) as integer,j as integer,w as long)
t=0
b=HIGHBYTE (w)
For k=0 to  3
```

```
    b= ftable( b xor keymatrix(j,k) )
    t = t xor ROTL(qbox (b) ,k)
next
return (b<<4 + t xor w)
```

To generate four S-box each of 256 entry and of size 32-bit:

```
 For j=0 to 3
    For i=0  to 255
S(i,j) = sbox(keymatrix,j, Clng(i<<8) xor (j<<8)))
Next
Next
```

4. Results and Discussion

In this section the proposed algorithm is evaluated and its advantages are discussed.

4.1. Evaluation methodology

Figure 4(a) shows the effect of nose shape on C_{D0} with cylindrical afterbody as a function of Mach number. The drag of cone-cylinder combination was the lowest at the considered Mach numbers. It is clear that the bluntness of nose causes the drag to increase.

The chosen factor to determine the performance of the proposed algorithm is the key generation speed for both versions and the algorithm's speed to encrypt/decrypt data blocks of various sizes.

The encryption time is considered the time that an encryption algorithm takes to produce the key plus the time to produce a cipher text from a plaintext. The decryption time is the time required to produce the key plus the time to produce the plaintext from the cipher text. Encryption time is used to calculate the throughput of an encryption scheme.

The throughput of the encryption scheme is calculated as the total plaintext in bytes encrypted divided by the encryption time. By considering different sizes of data blocks (1 kB to 15 kB) the algorithms were evaluated in terms of the time required to encrypt and decrypt the data block. For short we refer to the blowfish algorithm with the proposed key generation as B1 and the original algorithm as B2.

4.2. Description of the results

Table 1 shows the time required to generate the S-box from a 128-bit key converted to 4×4 keymatrix, P-array and initialization process. Table 2 Summarized the required time for encryption and decryption process for both algorithms and the average time for different input size. The encryption/decryption time for B1 and B2 are expressed as the sum of the key generation process and the encryption/decryption process. Finally, the throughput for each algorithm is computed.

Table 1. Time Required for the Proposed Key Generation Subprocess.

Process	Time in seconds (s)
Initialization	1
P-array generation	1
S-box generation	1
Total	3

Table 2. Time Required for Encryption and Decryption Process for Different File Size in Both Algorithms.

File Size in kB	Encryption Time B1/s	Encryption Time B2/s	Decryption Time B1/s	Decryption Time B2/s
1	7	16	7	16
2	8	17	8	17
4	15	24	16	25
8	26	35	27	36
15	63	72	63	72
Average Time	23.2	32.8	23.6	33.2
Throughput bytes/s	264.827	187.317	260.338	185.060

4.3. Discussion

The proposed algorithm makes the exhaustive key search attack and other types of attack which try to find the weakness in the design to leak some information about the initial value or secret value are difficult. The following summarizes the security analysis of the design:

- The values of nonce, Qbox, ftable are unknown. This makes the algebraic attack infeasible.
- The LFSR cycle length is equal to $(2^7\text{-}1) \times (2^9\text{-}1)$ and so on for the rest register. The tap function is XORing function between the cell 0 and cell n-1. The output from all the LFSRs are forwarded to the SRB function which is unknown and its output are balanced, and uncorrelated. The output of this function is the initial value of the Lr register. All these properties make it difficult to the guess and determination attack to guess the initial value of Lr register even the attack on the LFSR is possible.
- The multiple processes used to generate P-array are sufficient to eliminate any relation between the input stream and output key.

The proposed algorithm has the following advantages:

- The key and S-boxes are generated as needed not to be saved and this eliminate the need to preserve 4168 byte from the memory system to store them which results in applicability in smart card applications that have 5 kB memory.
- The algorithm complexity evaluation show that the subkeys generation process requires a total of 272 iterations to complete compared to 521 in old generation

process. Which is faster than the old process in software implementation as will be explained later. So the tradeoff between the memory size and the speed are eliminated and the subkeys are generated frequently without fair of slowing the algorithm. This makes the algorithm applicable in application requiring changing secret key frequently.

- It's required only to store Qbox and ftable, they can burn into ROM.

Finally, the design of the generator depends on SSS stream generator for some extent which make some of its advantages inherited like allowing for non-secret nonce.

5. Conclusions

In this paper a new key and S-box generation algorithm was proposed for blowfish block cipher. The proposed process makes use of key stream generation for SSS cipher system with some necessary modification. In this process, S-box generation process is no longer depending in P-array. The generation of S-box is based on the initial value generated at initialization step, so these changes complicated the analysis of P-array and S-box for their independence. The performance analysis of the proposed algorithm in term of time shows that the algorithm is faster than the old algorithm where the proposed algorithm requires a total of 272 iterations to generate the subkeys compared to 521 iterations. This give the advantages in both time and memory size where there is no need to store the subkeys and preserve approximately 4 kB of memory but it can be generated quickly

References

1. Schneier, B.(1996). *Applied cryptography second edition protocols*, *Algorithms*, *and Source codes in C*. John Wiley and Sons, Inc.
2. Schneier, B. (1994). Description of a new variable-length key, 64-bit block cipher (blowfish) fast software encryption. *Cambridge Security Workshop Proceedings*, 191-204.
3. Rose, G. (1998). A stream cipher based on linear feedback over GF(2^8). *Information Security and Privacy, Lecture Notes in Computer Science*, Volume 1438, 135-146.
4. Rose, G. (1998). SOBER: A stream cipher based on linear feedback over GF(28). *Unpubl ished report , QUALCO MM Australia.*
5. Hawkes, P; Paddon, M.; Rose, G.G.; and de Vries, M.W. (2004). *Primitive Specification for SSS*. Qualcomm Australia
6. Milad, A.A.; Muda, H.Z.; Muhamad Noh, Z.A.B.; and Algaet, M.A. (2012). Comparative study of performance in cryptography algorithms (Blowfish and Skipjack). *Journal of Computer Science*, 8(7), 1191-1197.
7. Landge, I.; Contractor, B.; Patel, A.; and Choudhary, R. (2012), Image encryption and decryption using blowfish algorithm. *World Journal of Science and Technology*, 2(3),151-156.

8. William, S. (2012). *Network security essentials: Applications and standards.* (5th Ed.), Prentice Hall, New Jersey.

Appendix A

Representation of the Results

Simulation results for encryption part of Table 2 are shown in Fig. A-1. Throughput for both algorithm in encryption and decryption are shown in Figs. A-2 and A-3 respectively.

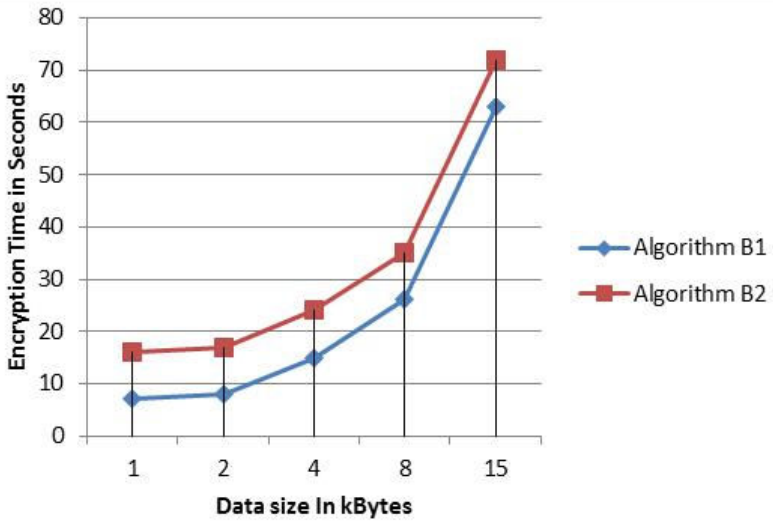

Fig. A-1. Time Consumption of Encryption Algorithm.

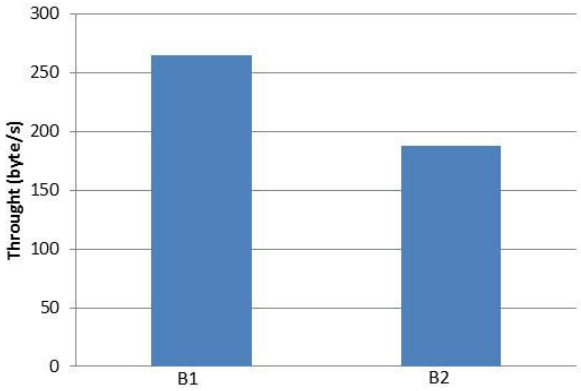

Fig. A-2. Throughput of each Encryption Algorithm (Byte/s).

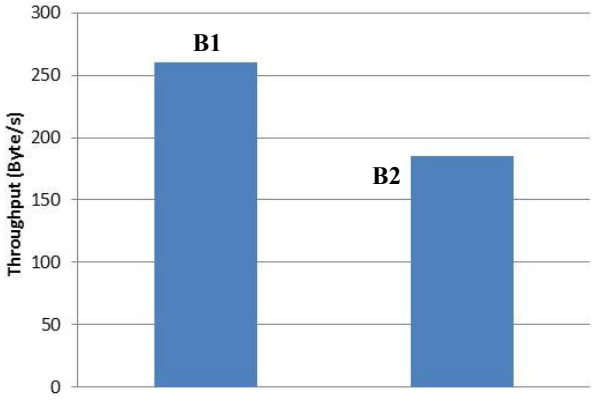

Fig. A-3. Throughput of Each Decryption Algorithm (Byte/s).

Appendix B

Computer Programme

The experiments are conducted using laptop with 2.5 GHz CPU, Intel Core i5 with 4GB of RAM. The simulation program is compiled using the default settings in .NET 2008 visual studio over Windows 2007.